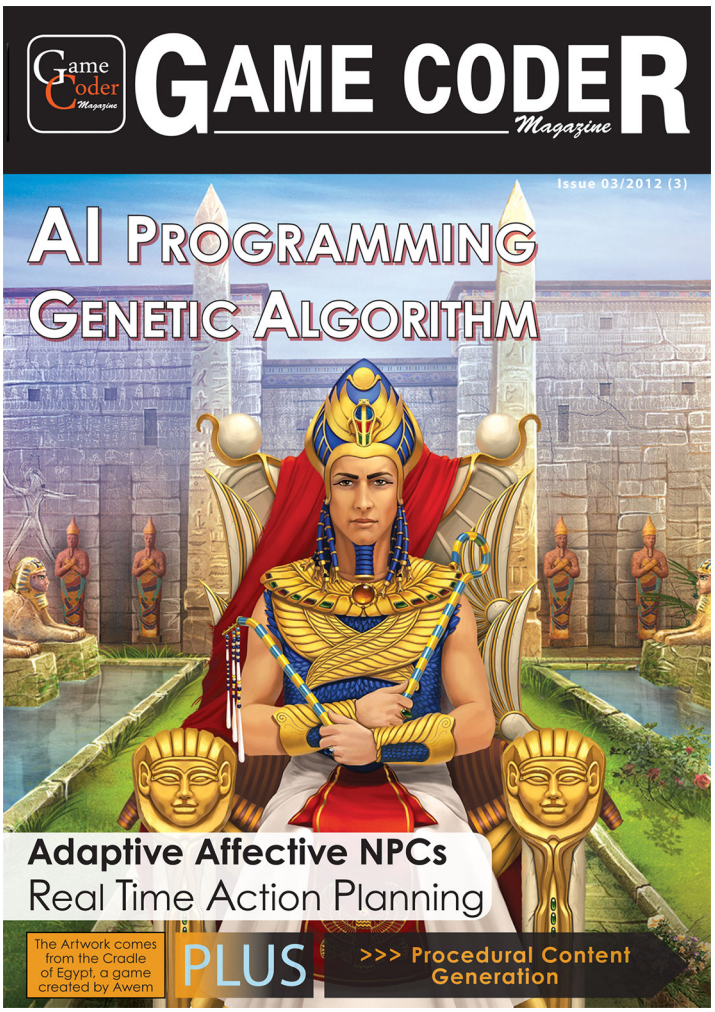

GameCoderMag_03_02-1.jpg

Creating Smart Enemies With Genetic Algorithms

by Sam Redfern

Genetic Algorithms (GA) are an approach to AI that makes uses of the 'survival of the fittest' concept, as seen in Darwinian evolution, in order to search for optimal (and sometimes novel) solutions to problems. They have been used in various games, most commonly to pre-calculate optimal actions for computer-controlled entities.

An interesting project by *van Galen Last* used *Quake 3* as the platform for first person shooter (FPS) AI, in which GAs were used to evolve bot strategies (see references at the end of this article). Several racing games, including my own game *Darkwind*, have used GAs to improve computer driving behaviour without having to resort to cheating (which is still unfortunately one of the most common forms of AI!) A small number of other games have used GAs (and related AI techniques) for real-time learning, whereby the computer agent actually learns during (rather than before) the game. In the case of *Black & White* and *Creatures*, the training of the bots actually *was* the game, more or less.

Certainly GAs are most valuable when operating on a very well defined problem, where the problem can be easily parametrised using a small set of variables, and where the 'fitness' of a particular genome can be accurately calculated.

Genetic Algorithms

To develop a GA, you first need a parameterised solution to your problem – this means, essentially, having a set of variables which define an approach to solving that problem, and the GA will find the best values for those variables to take.

Referring to figure 1, the first thing we need is a population of genes – in other words, a number of sets of possible values for the variables defining the solution. To keep things simple, this diagram illustrates an unrealistically small population of just 4 genes, each with a simple 6-bit genome. In some cases, we construct this initial population using random numbers for each

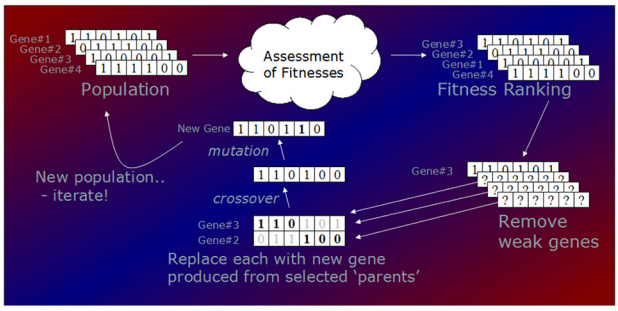

Figure 1: *The general operation of a Genetic Algorithm (GA)*

GameCoderMag_03_02-8.jpg

gene – although for more complex problems the algorithm may have trouble finding good solutions. In *Darkwind*, my initial population was based on a human-defined approximate solution.

The GA requires us to assess the fitness of each gene; this is entirely dependent on the problem we're trying to solve, and it's best if it's an accurate and objective measurement. In a racing game, laptimes are an ideal mesurement of fitness. In an FPS, a good measurement might be the ratio of kills to deaths achieved by a bot under the control of the GA. Having calculated the fitness of each gene, we sort the population by fitness, and remove some of the weakest ones.

After removing some of the weakest genes, we replace them with new genes: we then have a new population, ready to start the process again. The new genes are based on some from the surviving 'fit' population. The usual approach is to allow the fittest genes to have the greatest chance of being a 'parent' in the next generation; there are various ways of applying this rule.

The algorithm is likely to need lots of iterations, and therefore will ideally run in an 'unsupervised' way – i.e. without any need for human input. This may suit some games more than others, depending on whether a suitable fitness function can be devised which can operate without the need for human input. Again, consider an FPS: is it good enough to have bots fighting bots, or will this provide a poor challenge which doesn't lead to decent evolution? In a racing game, it's simpler since laptimes are an objective measurement of fitness which isn't related in any way to how an opponent is behaving – so a racing game is ideal for unsupervised learning.

One of the things that makes GAs different from related 'guided random search' techniques such as *simulated annealing* or *hill-climbing*, is that two parents (rather than just one) are selected to 'breed' each new gene. There are various ways of combining the genes of two parents, but the traditional approach is to (randomly) select a 'crossover' point and to create the new gene from the first part of one parent and the second part of the other. Various theoretical researchers have studied the effect of crossover (for example, see the report from Senaratna in the references) and it is generally observed that for many problems it is useful to have this form of 'sexual reproduction' since it gives the possibility of combining the best of one parent with the best of another.

Finally, each new gene is given a chance of 'mutation' – whereby one or more of its values are randomised. This is an important part of the algorithm as it provides a way for new features to get introduced into a population. Just like in real life evolution, most mutations will lead to an unfit gene and will therefore disappear on the next iteration; but occasionally a mutation will lead to superior fitness and therefore the mutated value will rapidly become a standard part of the population.

There are various aspects of the GA that need to be decided upon, and perhaps experimented with, for a given problem. How large should the population be? How do we determine crossover? How do we select parents? How common should mutation be? Each of these aspects can have a strong effect on how well the GA is able to converge on an optimal solution, or indeed whether it can do so at all within a reasonable number of iterations. Although many people have written about these aspects of GAs, it is generally the case that each problem is unique and therefore it comes down to trial and error when defining what's best for your own problem.

As you can see, the operation of a genetic algorithm is actually quite simple. Whether it can achieve the desired results depends, of course, on how well your problem has been parameterised, and how well bounded the problem, as stated, is. For example, it would be foolhardy to expect a bot to learn when to turn left, right, run forward and fire in a first person shooter (FPS) simply by ranking its fitness. Like most problems in AI, the trick is to develop techniques that operate at the right level of abstraction. In a FPS, this might mean using a multi-layer AI approach, perhaps using GAs for the high level strategic decision making (run away, hide, charge), and using other techniques such as A* pathfinding for the low-level operations such as movement.

Breeding Race Drivers

To illustrate GAs in more detail, I will use my own work on the online racing/combat game *Darkwind*. Darkwind supports a detailed physical driving model and provides a set of demanding off-road racetracks. Since it is an online game with hundreds of currently active players, and in which upwards of 1000 separate races are run every week, Darkwind also provides an excellent opportunity to compare computer drivers with expert human players.

During a race, the computer drivers use a waypoint approach: essentially, each racetrack has a number (20-60) of waypoints, and the computer steers the cars towards each in turn. Each rectangular waypoint consists of not only a position, but also a target speed and a size. As the boundary of a waypoint is crossed, the bot switches to the next one – this provides enough subtlety for quite specific steering actions to be developed for particular parts of a racetrack. However, it also means that manually defining (and especially tweaking/optimising) these waypoints is time-consuming: this is where GAs are very useful. See figure 2 for an illustration of a racetrack with waypoints overlaid.

From my own perspective therefore, one of the main motivations for this work was to reduce the development effort required to produce effective drivers: the GA approach is used in its classic optimisation role. It was also a goal to efficiently produce a database of interesting and varied sets of waypoints that can be used to control different cars in a race, making the computer-controlled cars act less predictably. By varying the fitness function of the GA, I was also able to produce behaviours that are optimised in different ways, and that therefore exhibit different driving styles (e.g. cautious versus reckless).

In Darkwind, a sophisticated physical model of vehicle dynamics has been employed: this includes an approximation of vehicle aerodynamics (downforce, air resistance and slipstreaming), variable tyre characteristics (lateral and longitudinal deformations, static and kinetic friction, performance degradation and damage), suspension (length, spring forces), engine performance curves, and rigid-body collision resolution. In addition, the racetracks employ a number of types of surface with varying friction and solidity, and contain not only obstacles but also in some cases quite varied terrain formations. The complexity of this system provides a challenging game with keenly contested races and on-line lap records: on many tracks, the emphasis is on maintaining a good velocity while cornering, or on attacking a terrain feature such as a jump in an optimal fashion. On other tracks the emphasis may involve minimizing tyre-wear. This complexity means that creating effective and interesting computer drivers through scripted calculations or algorithmically defined racing lines is infeasible.

Figure 2 illustrates a set of waypoints on a tight, fast corner, which has concrete blocks at its apex. Waypoint 8 encourages the car to approach the corner from the outside of the track, and to turn into the corner early and gently (aiming for waypoint 9), thereby minimising skidding and maintaining a good speed. The concrete blocks are avoided by the influence of waypoint 10; this waypoint also encourages the car to hit an ideal speed of 96mph, which proves low enough for this specific car with these specific tyres to exit the corner without crossing from the tarmac surface to the sandy margin and concrete barriers on the right. Waypoints 11, 12 and 13 encourage the car to stay on the tarmac, and the car exits on the ideal side of the road for the subsequent straight section and left-turning corner (out of picture). The final exit speed of 78mph is considered almost-optimal for this vehicle.

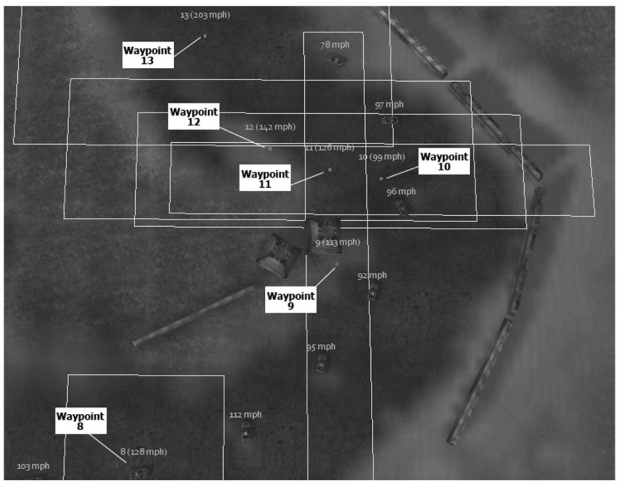

Figure 2: *An evolved set of racing waypoints on a dusty tarmac surface*

GameCoderMag_03_02-9.jpg

GameCoderMag_03_02-10.jpg

The car aims for the centre of each waypoint (numbered 8 through 13 in figure 2) while targeting the waypoint's indicated speed. As soon as the boundary of a waypoint is crossed, the proceeding waypoint becomes the target. The position, size, and target speeds of the waypoints are evolved using a GA. The position and speed of a car over a period of 8 seconds of the game is also illustrated.

Encoding the Genome

In Darkwind, each genome consists of a set of waypoints stored as real numbers. The data for each waypoint consists of: centre position (x, y), dimensions (x, y), and target speed. Depending on the size and complexity of the racetrack, the number of waypoints used may vary from about 20 to about 60. All genomes in a population contain the same number of waypoints.

Each candidate genome was tested 10 times, with the first lap in each sequence disregarded so as to allow the cars to achieve optimum speed by the start of a timed lap. Collision damage was ignored during training, although tyre-wear due to excessive skidding or driving over rocky ground was not. Even without damage, a car suffering a collision will obtain a very poor laptime due to lost momentum, spinning and so on – I did not wish to further penalise all subsequent laps by damaging the car's physical condition.

The best ranking 50% of the population was copied unaltered to the proceeding generation, while the remaining 50% was discarded and replaced with new genomes. Parents for each new genome were selected from the elite 50%, with a bias towards the highest ranked. Crossover was chosen at a random point in the gene, but only where the following additional crossover compatibility test is passed.

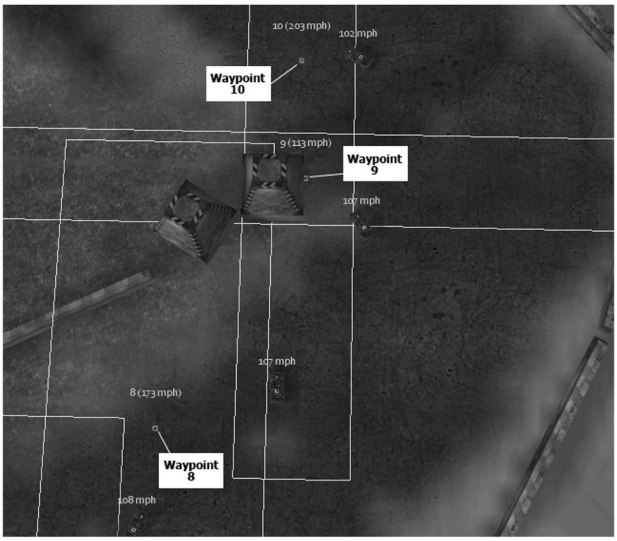

Figure 3: *The same corner is depicted as in figure 2. In this case, the best rather than average laptime has been used as the fitness function, yielding a more risk-taking driving style.*

GameCoderMag_03_02-11.jpg

For a crossover point at waypoint x to be valid for parents p1 and p2, where p1[x] represents waypoint x of parent p1:

- The distance from p1[x-2] to p2[x] must be larger than the distance from p1[x-1] to p2[x]
- The distance from p1[x-1] to p2[x+1] must be larger than the distance from p2[x] to p2[x+1]

This test ensures that the crossover point does not cause the car to turn around on the racetrack and start driving in the wrong direction, which would clearly be a bad idea and would lead to much slower learning rates.

Overall Performance

A primary performance measurement involves the assessment of laptimes achieved by evolved sets of waypoints versus the original manually defined waypoints deployed by the racetrack designers. Having experimented with various mutation schema, numbers of waypoints, fitness functions and population sizes, an improvement of between 8% and 30% in average laptime over 10 laps has been observed on all racetracks and vehicle types.

The most effective GA parameters for optimising on each track and vehicle do vary somewhat, but this typically affects the efficiency of search (i.e., training time) rather than the final result. The computer has not yet beaten the best laptimes recorded by human players over the past 4 years. It should be noted that in many cases, close to 100,000 laps have been attempted by human players on each racetrack.

Influencing Driving Styles

By varying the fitness function, we can evolve driving styles varying from cautious (by defining fitness as the worst laptime of the 10 laps recorded), to optimal on average (by using the average laptime), to risk-taking (by using the best laptime). This allows different personalities to be presented in the game, and perhaps for some race-time strategy to be attempted, depending on the current state of the race. These sorts of optimsations can certainly contribute to making more interesting and believable computer opponents in games.

Figure 3 illustrates the same corner as that shown in figure 2. The fitness function has been changed from average laptime to best laptime, and therefore a more aggressive racing line has evolved, which encourages the car to drive closer to the dangerous apex and hold a higher speed, risking tyre-wear and spin-outs, yet performing better when successful, and yielding a higher exit speed from the corner.

Evolving Expert Knowledge

Most of the racetracks in the game have a number of critical features where the success or failure of a race is often decided. Only the leading human players have obtained mastery of these features, and certainly without GA optimisation it is very unlikely that computer drivers will perform well on them. In the following sub-sections I explore a few racetracks on which experiments have been run.

Dirt Racing Track

This is a dusty, bumpy track with many elevation changes. The final straight leading to the finishing line is a steep, bumpy hill with an adverse camber, which tends to tip the car outwards into a collision course with the finishing gate. Compensating for this often causes computer drivers and novice players to oversteer or spin their car and lose their momentum. The evolved waypoints encouraged the computer driver to stay close to the left side of the track (which is the higher ground) and to avoid straying onto the sloping part of the road. This was achieved through use of a large number of tightly packed waypoints close to the left hand edge.

Halfway around this track is a large bump with a pit on one side. Taking this section badly typically means a higher jump, heavier landing, and a substantial loss of momentum. The evolved computer drivers were found to swing wide before the bump and then attack its low-point at an approximately 45 degree angle: this minimizes the height that the car jumps, and maximizes its exit speed. This is precisely the same racing line that most expert human drivers use.

On this circuit, the best laptime recorded using the evolved racing lines with a muscle-car was 32.03 seconds, which is a 10.44 seconds (24.6%) improvement on the developer-defined waypoints. The severely bumpy/hilly nature of this track made the original waypoints generalize poorly for the various vehicles – the efficiency of using GAs to produce per-vehicle optimised versions is clear. The best ever human-recorded laptime using the same vehicle is 30.87 seconds.

Northern Speedway

This is a broadly circular circuit with a very loose surface (desert sand) – it's an easy track to drive but a difficult one to drive optimally. On this circuit, the evolved racing lines included power-slides (drifts) on the two tightest bends – the waypoints were found to be positioned several metres inside the apex, so that the cars used understeering to their advantage to navigate the corners at maximal speed while staying close to the inside of the track.

Overall, the best time achieved by the computer when controlling a fast vehicle was 24.01 seconds, a reduction of 2.67 seconds (9.9%) when compared with the time achieved using the original, developer-defined waypoints. The best-ever human laptime on this track using the same car is 21.69 seconds.

Northern Foothills Racetrack

This is a tight, fairly smooth racetrack with sharp corners. The centre of the track has a good surface but

GameCoderMag_03_02-12.jpg

this is typically quite narrow and the margins are sandy – it is very detrimental to a laptime if a car strays onto the margins. The computer-controlled cars used to be notoriously ineffective on this circuit, due to the difficulty experienced during development in manually defining an optimal set of waypoints.

On this circuit, the best laptime recorded using the evolved racing lines with a road-car was 56.93 seconds, which is an 11.98 seconds (17.4%) improvement on the developer-defined waypoints. The best ever human-recorded laptime is 53.56 seconds.

One hairpin corner in particular is difficult to drive optimally due to its sandy surface, and can be exited at above 50mph if taken well and if the narrow tarmac track-centre is successfully targeted on the exit. Taken badly, a very low exit speed or a spin-out are common. Figure 4 illustrates an evolved set of waypoints being used in-game on this corner. To hit the narrow tarmac centre-line on the exit is difficult at high speed since the corner itself is so severe and surfaced with sand. The evolved waypoints succeed in obtaining an exit speed of 45mph in this sample run. Note that the strangely-positioned waypoint 12 is 'junk' data that never affects the racing line due to its overlap with waypoint 11. (In real life, all creatures carry a lot of 'junk DNA' too).

Genetic Algorithms Are Not The Entire Answer

I would like to conclude by emphasising that usually you won't produce useful AI by using GAs alone. What is important is that you understand what GAs are good at, and to incorporate them appropriately into a wider AI strategy. In the case of Darkwind, during a race the AI drivers attempt to steer towards their next waypoint, however at the same time they also monitor nearby car positions and velocities as well as nearby obstacles. If necessary, collision avoidance becomes an 'interrupt condition' which over-rides the underlying goal of steering towards the next waypoint.

Another factor which I have not yet incorporated, but which I intend to, is the current speed of the AI car: my GAs evolve ideal waypoints for cars running at high speed, but sometimes this is entirely inappropriate for cars which are running slower – the racing line may for example assume a corner is being skidded around, and may cause a slower vehicle to drive straight into an obstacle.

Sam Redfern

Sam Redfern is a university lecturer in Ireland, holding an M.Sc. and Ph.D. in Information Technology. He researches and teaches graphics and games programming, as well as virtual collaboration software and A.I. He also develops games, operating the indie studio Psychic Software in his spare time.

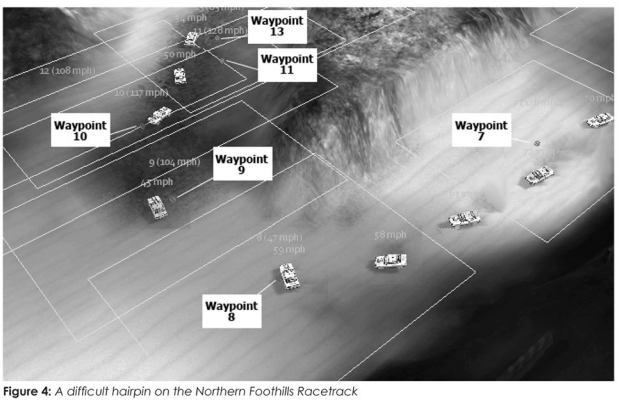

Figure 4: *A difficult hairpin on the Northern Foothills Racetrack*

GameCoderMag_03_02-13.jpg